Package 'asa'

January 6, 2026

Title AI Search Agent for Large-Scale Research Automation

Version 0.1.0

Description Provides an LLM-powered research agent for performing AI search tasks at large scales. Uses a ReAct (Reasoning + Acting) agent pattern with web search capabilities via DuckDuckGo and Wikipedia. Implements DeepAgent-style memory folding for context management. The agent is built on 'LangGraph' and supports multiple LLM backends including 'OpenAI', 'Groq', and 'xAI'.

URL `https://github.com/cjerzak/asa-software`

BugReports `https://github.com/cjerzak/asa-software/issues`

Depends R (>= 4.0.0)

License GPL-3

Encoding UTF-8

LazyData false

Imports reticulate (>= 1.28),
jsonlite,
rlang,
digest,
processx

Suggests testthat (>= 3.0.0),
knitr,
rmarkdown,
future,
future.apply

VignetteBuilder knitr

RoxygenNote 7.3.3

Config/testthat/edition 3

SystemRequirements Python (>= 3.11), Conda, Tor (optional, for anonymous searching)

Contents

as.data.frame.asa_audit_result
Convert asa_audit_result to Data Frame

Description

Convert asa_audit_result to Data Frame

Usage

```
## S3 method for class 'asa_audit_result'
as.data.frame(x, ...)
```

Arguments

`x`	An asa_audit_result object
`...`	Additional arguments (ignored)

Value

The audited data.frame with audit columns

as.data.frame.asa_enumerate_result
Convert asa_enumerate_result to Data Frame

Description

Convert asa_enumerate_result to Data Frame

Usage

```
## S3 method for class 'asa_enumerate_result'
as.data.frame(x, ...)
```

Arguments

`x`	An asa_enumerate_result object
`...`	Additional arguments (ignored)

Value

The data data.frame from the result

as.data.frame.asa_result — *Convert asa_result to Data Frame*

Description

Convert asa_result to Data Frame

Usage

```
## S3 method for class 'asa_result'
as.data.frame(x, ...)
```

Arguments

`x`	An asa_result object
`...`	Additional arguments (ignored)

Value

A single-row data frame

asa_agent — *Constructor for asa_agent Objects*

Description

Creates an S3 object representing an initialized ASA search agent.

Usage

```
asa_agent(python_agent, backend, model, config, llm = NULL, tools = NULL)
```

Arguments

`python_agent`	The underlying Python agent object
`backend`	LLM backend name (e.g., "openai", "groq")
`model`	Model identifier
`config`	Agent configuration list
`llm`	Optional LLM object used by LangGraph
`tools`	Optional list of tools associated with the agent

Value

An object of class `asa_agent`

asa_audit — *Audit Enumeration Results for Completeness and Quality*

Description

Validates enumeration results for completeness, consistency, and data quality using either Claude Code (CLI) or a LangGraph-based audit pipeline.

Usage

```
asa_audit(
  result,
  query = NULL,
  known_universe = NULL,
  checks = c("completeness", "consistency", "gaps", "anomalies"),
  backend = c("claude_code", "langgraph"),
  claude_model = "claude-sonnet-4-20250514",
  llm_model = "gpt-4.1-mini",
  interactive = FALSE,
  confidence_threshold = 0.8,
  timeout = 120,
  verbose = TRUE,
  agent = NULL
)
```

Arguments

`result`	An `asa_enumerate_result` object or a data.frame to audit
`query`	The original enumeration query (inferred from result if NULL)
`known_universe`	Optional vector of expected items for completeness check
`checks`	Character vector of checks to perform. Options: "completeness", "consistency", "gaps", "anomalies". Default runs all checks.
`backend`	Backend to use for auditing: "claude_code" (CLI) or "langgraph"
`claude_model`	Model to use with Claude Code backend
`llm_model`	Model to use with LangGraph backend
`interactive`	If TRUE and using claude_code backend, spawn an interactive Claude Code session instead of programmatic invocation
`confidence_threshold`	Flag items with confidence below this threshold
`timeout`	Timeout in seconds for the audit operation
`verbose`	Print progress messages
`agent`	Existing asa_agent for LangGraph backend (optional)

Details

The audit function adds three columns to the data:

- `_audit_flag`: "ok", "warning", or "suspect"
- `_audit_notes`: Explanation of any issues
- `_confidence_adjusted`: Revised confidence after audit

## Audit Checks

completeness: Checks for missing items by comparing against known_universe (if provided) or using domain knowledge.

consistency: Validates data types, patterns, and value ranges.

gaps: Identifies systematic patterns of missing data (geographic, temporal, categorical gaps).

anomalies: Detects duplicates, outliers, and suspicious patterns.

Value

An `asa_audit_result` object containing:

`data`	Original data with audit columns added (_audit_flag, _audit_notes)
`audit_summary`	High-level summary of findings
`issues`	List of identified issues with severity and descriptions
`recommendations`	Suggested remediation queries
`completeness_score`	0-1 score for data completeness
`consistency_score`	0-1 score for data consistency

Examples

```
## Not run:
# Audit enumeration results with Claude Code
senators <- asa_enumerate(
  query = "Find all current US senators",
  schema = c(name = "character", state = "character", party = "character")
)
audit <- asa_audit(senators, backend = "claude_code")
print(audit)

# Audit with known universe for precise completeness check
audit <- asa_audit(senators, known_universe = state.abb)

# Interactive mode for complex audits
asa_audit(senators, backend = "claude_code", interactive = TRUE)

# Use LangGraph backend
audit <- asa_audit(senators, backend = "langgraph", agent = agent)

## End(Not run)
```

`asa_audit_result` *Constructor for asa_audit_result Objects*

Description

Creates an S3 object representing the result of a data quality audit.

Usage

```
asa_audit_result(
  data,
  audit_summary,
  issues,
  recommendations,
  completeness_score,
  consistency_score,
  backend_used,
  elapsed_time,
  query = NULL,
  checks = NULL
)
```

Arguments

`data`	data.frame with original data plus audit columns (_audit_flag, _audit_notes)
`audit_summary`	Character string with high-level findings
`issues`	List of identified issues with severity and descriptions
`recommendations`	Character vector of suggested remediation queries
`completeness_score`	Numeric 0-1 score for data completeness
`consistency_score`	Numeric 0-1 score for data consistency
`backend_used`	Which backend performed the audit ("claude_code" or "langgraph")
`elapsed_time`	Execution time in seconds
`query`	The original query (if available)
`checks`	Character vector of checks that were performed

Value

An object of class `asa_audit_result`

`asa_config` *Create ASA Configuration Object*

Description

Creates a configuration object that encapsulates all settings for ASA tasks. This provides a unified way to configure backend, model, search, temporal, and resource settings in a single object.

Usage

```
asa_config(
  backend = NULL,
  model = NULL,
  conda_env = NULL,
  proxy = NULL,
  workers = NULL,
  timeout = NULL,
  rate_limit = NULL,
  memory_folding = NULL,
  memory_threshold = NULL,
  memory_keep_recent = NULL,
  temporal = NULL,
  search = NULL
)
```

Arguments

`backend`	LLM backend: "openai", "groq", "xai", "exo", "openrouter"
`model`	Model identifier (e.g., "gpt-4.1-mini")
`conda_env`	Conda environment name (default: "asa_env")
`proxy`	SOCKS5 proxy URL or NULL to disable
`workers`	Number of parallel workers for batch operations
`timeout`	Request timeout in seconds
`rate_limit`	Requests per second
`memory_folding`	Enable DeepAgent-style memory folding
`memory_threshold`	Messages before folding triggers
`memory_keep_recent`	Messages to preserve after folding
`temporal`	Temporal filtering options (use `temporal_options()`)
`search`	Search configuration (use `search_options()`)

Details

The configuration object can be passed to `run_task()`, `run_task_batch()`, `asa_enumerate()`, and other functions to provide consistent settings across operations.

Value

An object of class asa_config

See Also

temporal_options, search_options

Examples

```
## Not run:
# Create configuration
config <- asa_config(
  backend = "openai",
  model = "gpt-4.1-mini",
  workers = 4,
  temporal = temporal_options(time_filter = "y")
)

# Use with run_task
result <- run_task(prompt, config = config)

## End(Not run)
```

---

asa_enumerate *Multi-Agent Research for Open-Ended Queries*

Description

Performs intelligent open-ended research tasks using multi-agent orchestration. Decomposes complex queries into sub-tasks, executes parallel searches, and aggregates results into structured output (data.frame, CSV, or JSON).

Usage

```
asa_enumerate(
  query,
  schema = NULL,
  output = c("data.frame", "csv", "json"),
  workers = NULL,
  max_rounds = NULL,
  budget = list(queries = 50L, tokens = 200000L, time_sec = 300L),
 stop_policy = list(target_items = NULL, plateau_rounds = 2L, novelty_min = 0.05,
    novelty_window = 20L),
  sources = list(web = TRUE, wikipedia = TRUE, wikidata = TRUE),
  temporal = NULL,
  pagination = TRUE,
  progress = TRUE,
  include_provenance = FALSE,
  checkpoint = TRUE,
  checkpoint_dir = tempdir(),
  resume_from = NULL,
  agent = NULL,
  backend = NULL,
  model = NULL,
  conda_env = NULL,
  verbose = TRUE
)
```

Arguments

`query`	Character string describing the research goal. Examples: "Find all current US senators with their state, party, and term end date"
`schema`	Named character vector defining the output schema. Names are column names, values are R types ("character", "numeric", "logical"). Use NULL or "auto" for LLM-proposed schema.
`output`	Output format: "data.frame" (default), "csv", or "json".
`workers`	Number of parallel search workers. Defaults to value from `ASA_DEFAULT_WORKERS` (typically 4).
`max_rounds`	Maximum research iterations. Defaults to value from `ASA_DEFAULT_MAX_ROUNDS` (typically 8).
`budget`	Named list with resource limits: • queries: Maximum search queries (default: 50) • tokens: Maximum LLM tokens (default: 200000) • time_sec: Maximum execution time in seconds (default: 300)
`stop_policy`	Named list with stopping criteria: • target_items: Stop when this many items found (NULL = unknown) • plateau_rounds: Stop after N rounds with no new items (default: 2) • novelty_min: Minimum new items ratio per round (default: 0.05) • novelty_window: Window size for novelty calculation (default: 20)
`sources`	Named list controlling which sources to use: • web: Use DuckDuckGo web search (default: TRUE) • wikipedia: Use Wikipedia (default: TRUE) • wikidata: Use Wikidata SPARQL for authoritative enumerations (default: TRUE)
`temporal`	Named list for temporal filtering: • after: ISO 8601 date string (e.g., "2020-01-01") - results after this date • before: ISO 8601 date string (e.g., "2024-01-01") - results before this date • time_filter: DuckDuckGo time filter ("d", "w", "m", "y") for day/week/month/year • strictness: "best_effort" (default) or "strict" (verifies dates via metadata) • use_wayback: Use Wayback Machine for strict pre-date guarantees (default: FALSE)
`pagination`	Enable pagination for large result sets (default: TRUE).
`progress`	Show progress bar and status updates (default: TRUE).
`include_provenance`	Include source URLs and confidence per row (default: FALSE).
`checkpoint`	Enable auto-save after each round (default: TRUE).
`checkpoint_dir`	Directory for checkpoint files (default: tempdir()).

`resume_from`	Path to checkpoint file to resume from (default: NULL).
`agent`	An initialized `asa_agent` object. If NULL, uses the current agent or creates a new one with specified backend/model.
`backend`	LLM backend if creating new agent: "openai", "groq", "xai", "openrouter".
`model`	Model identifier if creating new agent.
`conda_env`	Conda environment name (default: "asa_env").
`verbose`	Print status messages (default: TRUE).

Details

The function uses a multi-agent architecture:

1. **Planner**: Decomposes query into facets and identifies authoritative sources
2. **Dispatcher**: Spawns parallel workers for each facet
3. **Workers**: Execute searches using DDG, Wikipedia, and Wikidata
4. **Extractor**: Normalizes results to match schema
5. **Deduper**: Removes duplicates using hash + fuzzy matching
6. **Stopper**: Evaluates stopping criteria (novelty, budget, saturation)

For known entity types (US senators, countries, Fortune 500), Wikidata provides authoritative enumerations with complete, verified data.

Value

An object of class `asa_enumerate_result` containing:

- data: data.frame with results matching the schema
- status: "complete", "partial", or "failed"
- stop_reason: Why the search stopped
- metrics: List with rounds, queries_used, novelty_curve, coverage
- provenance: If include_provenance=TRUE, source info per row
- checkpoint_file: Path to checkpoint if saved

Checkpointing

With checkpoint=TRUE, state is saved after each round. If interrupted, use resume_from to continue from the last checkpoint:

```
result <- asa_enumerate(query, resume_from = "/path/to/checkpoint.rds")
```

Schema

The schema defines expected output columns:

```
schema = c(name = "character", state = "character", party = "character")
```

With schema = "auto", the planner agent proposes a schema based on the query.

See Also

`run_task`, `initialize_agent`

Examples

```
## Not run:
# Find all US senators
senators <- asa_enumerate(
  query = "Find all current US senators with state, party, and term end date",
  schema = c(name = "character", state = "character",
             party = "character", term_end = "character"),
  stop_policy = list(target_items = 100),
  include_provenance = TRUE
)
head(senators$data)

# Find countries with auto schema
countries <- asa_enumerate(
  query = "Find all countries with their capitals and populations",
  schema = "auto",
  output = "csv"
)

# Resume from checkpoint
result <- asa_enumerate(
  query = "Find Fortune 500 CEOs",
  resume_from = "/tmp/asa_enumerate_abc123.rds"
)

# Temporal filtering: results from specific date range
companies_2020s <- asa_enumerate(
  query = "Find tech companies founded recently",
  temporal = list(
    after = "2020-01-01",
    before = "2024-01-01",
    strictness = "best_effort"
  )
)

# Temporal filtering: past year with DuckDuckGo time filter
recent_news <- asa_enumerate(
  query = "Find AI research breakthroughs",
  temporal = list(
    time_filter = "y"  # past year
  )
)

# Strict temporal filtering with Wayback Machine
historical <- asa_enumerate(
  query = "Find Fortune 500 companies",
  temporal = list(
    before = "2015-01-01",
    strictness = "strict",
    use_wayback = TRUE
  )
)

## End(Not run)
```

`asa_enumerate_result` *Constructor for asa_enumerate_result Objects*

Description

Creates an S3 object representing the result of an enumeration task.

Usage

```
asa_enumerate_result(
  data,
  status,
  stop_reason,
  metrics,
  provenance = NULL,
  plan = NULL,
  checkpoint_file = NULL,
  query = NULL,
  schema = NULL
)
```

Arguments

`data`	data.frame containing the enumeration results
`status`	Result status: "complete", "partial", or "failed"
`stop_reason`	Why the enumeration stopped (e.g., "target_reached", "novelty_plateau")
`metrics`	List with execution metrics (rounds, queries_used, etc.)
`provenance`	Optional data.frame with source information per row
`plan`	The enumeration plan from the planner agent
`checkpoint_file`	Path to saved checkpoint file
`query`	The original enumeration query
`schema`	The schema used for extraction

Value

An object of class `asa_enumerate_result`

`asa_response` *Constructor for asa_response Objects*

Description

Creates an S3 object representing an agent response.

Usage

```
asa_response(
  message,
  status_code,
  raw_response,
  trace,
  elapsed_time,
  fold_count,
  prompt
)
```

Arguments

`message`	The final response text
`status_code`	Status code (200 = success, 100 = error)
`raw_response`	The full Python response object
`trace`	Full text trace of agent execution
`elapsed_time`	Execution time in minutes
`fold_count`	Number of memory folds performed
`prompt`	The original prompt

Value

An object of class `asa_response`

`asa_result` *Constructor for asa_result Objects*

Description

Creates an S3 object representing the result of a research task.

Usage

```
asa_result(prompt, message, parsed, raw_output, elapsed_time, status)
```

Arguments

`prompt`	The original prompt
`message`	The agent's response text
`parsed`	Parsed output (list or NULL)
`raw_output`	Full agent trace
`elapsed_time`	Execution time in minutes
`status`	Status ("success" or "error")

Value

An object of class `asa_result`

`build_backend` *Build the Python Backend Environment*

Description

Creates a conda environment with all required Python dependencies for the asa search agent, including LangChain, LangGraph, and search tools.

Usage

```
build_backend(conda_env = "asa_env", conda = "auto", python_version = "3.13")
```

Arguments

`conda_env`	Name of the conda environment (default: "asa_env")
`conda`	Path to conda executable (default: "auto")
`python_version`	Python version to use (default: "3.13")

Details

This function creates a new conda environment and installs the following Python packages:

- langchain_groq, langchain_community, langchain_openai
- langgraph
- ddgs (DuckDuckGo search)
- selenium, primp (browser automation)
- beautifulsoup4, requests
- fake_headers, httpx
- pysocks, socksio (proxy support)

Value

Invisibly returns NULL; called for side effects.

Examples

```
## Not run:
# Create the default environment
build_backend()

# Create with a custom name
build_backend(conda_env = "my_asa_env")

## End(Not run)
```

build_prompt *Build a Task Prompt from Template*

Description

Creates a formatted prompt by substituting variables into a template.

Usage

```
build_prompt(template, ...)
```

Arguments

`template`	A character string with placeholders in the form `{variable_name}`
`...`	Named arguments to substitute into the template

Value

A formatted prompt string

Examples

```
## Not run:
prompt <- build_prompt(
  template = "Find information about {{name}} in {{country}} during {{year}}",
  name = "Marie Curie",
  country = "France",
  year = 1903
)

## End(Not run)
```

check_backend *Check Python Environment Availability*

Description

Checks if the required Python environment and packages are available.

Usage

```
check_backend(conda_env = "asa_env")
```

Arguments

`conda_env`	Name of the conda environment to check

Value

A list with components:

- available: Logical, TRUE if environment is ready
- conda_env: Name of the environment checked
- python_version: Python version if available
- missing_packages: Character vector of missing packages (if any)

Examples

```
## Not run:
status <- check_backend()
if (!status$available) {
  build_backend()
}

## End(Not run)
```

`configure_search` *Configure Python Search Parameters*

Description

Sets global configuration values for the Python search module. These values control timeouts, retry behavior, and result limits.

Usage

```
configure_search(
  max_results = NULL,
  timeout = NULL,
  max_retries = NULL,
  retry_delay = NULL,
  backoff_multiplier = NULL,
  captcha_backoff_base = NULL,
  page_load_wait = NULL,
  inter_search_delay = NULL,
  conda_env = "asa_env"
)
```

Arguments

`max_results`	Maximum number of search results to return (default: 10)
`timeout`	HTTP request timeout in seconds (default: 15)
`max_retries`	Maximum retry attempts on failure (default: 3)
`retry_delay`	Initial delay between retries in seconds (default: 2)
`backoff_multiplier`	Multiplier for exponential backoff (default: 1.5)

`captcha_backoff_base`	Base multiplier for CAPTCHA backoff (default: 3)
`page_load_wait`	Wait time after page load in seconds (default: 2)
`inter_search_delay`	Delay between consecutive searches in seconds (default: 0.5)
`conda_env`	Name of the conda environment (default: "asa_env")

Value

Invisibly returns a list with the current configuration

Examples

```
## Not run:
# Increase timeout for slow connections
configure_search(timeout = 30, max_retries = 5)

# Get more results
configure_search(max_results = 20)

# Add delay between searches to avoid rate limiting
configure_search(inter_search_delay = 2.0)

## End(Not run)
```

`configure_search_logging` *Configure Python Search Logging Level*

Description

Sets the logging level for the Python search module. This controls how much diagnostic output is produced during web searches.

Usage

```
configure_search_logging(level = "WARNING", conda_env = "asa_env")
```

Arguments

`level`	Log level: "DEBUG", "INFO", "WARNING" (default), "ERROR", or "CRITICAL"
`conda_env`	Name of the conda environment (default: "asa_env")

Details

Log levels from most to least verbose:

- DEBUG: Detailed diagnostic information for debugging
- INFO: General operational information
- WARNING: Indicates something unexpected but not an error (default)
- ERROR: Serious problems that prevented an operation
- CRITICAL: Very serious errors

Value

Invisibly returns the current logging level

Examples

```
## Not run:
# Enable verbose debugging output
configure_search_logging("DEBUG")

# Run a search (will show detailed logs)
result <- run_task("What is the population of Tokyo?", agent = agent)

# Disable verbose output
configure_search_logging("WARNING")

## End(Not run)
```

`configure_temporal` *Configure Temporal Filtering for Search*

Description

Sets or clears temporal filtering on the DuckDuckGo search tool. This affects all subsequent searches until changed or cleared.

Usage

```
configure_temporal(time_filter = NULL)
```

Arguments

`time_filter`	DuckDuckGo time filter: "d" (day), "w" (week), "m" (month), "y" (year), or NULL/NA/"none" to clear

Details

This function modifies the search tool's time parameter, which is passed to DuckDuckGo as the `df` parameter. The filter restricts results to content indexed within the specified time period.

Note: This only affects DuckDuckGo searches. For Wikidata queries with temporal filtering, use `asa_enumerate()` with its `temporal` parameter.

Value

Invisibly returns the previous time filter setting

Time Filter Values

- "d": Past 24 hours (day)
- "w": Past 7 days (week)
- "m": Past 30 days (month)
- "y": Past 365 days (year)
- NULL, NA, or "none": No time restriction (default)

See Also

run_task, asa_enumerate

Examples

```
## Not run:
# Restrict to past year
configure_temporal("y")
result <- run_task("Find recent AI breakthroughs", agent = agent)

# Clear temporal filter
configure_temporal(NULL)

# Past week only
configure_temporal("w")

## End(Not run)
```

extract_agent_results *Extract Structured Data from Agent Traces*

Description

Parses raw agent output to extract search snippets, Wikipedia content, URLs, JSON data, and search tier information. This is the main function for post-processing agent traces.

Usage

```
extract_agent_results(raw_output)
```

Arguments

raw_output	Raw output string from agent invocation (the trace field from an asa_response object)

Value

A list with components:

- search_snippets: Character vector of search result content
- search_urls: Character vector of URLs from search results
- wikipedia_snippets: Character vector of Wikipedia content
- json_data: Extracted JSON data as a list (if present)
- search_tiers: Character vector of unique search tiers used (e.g., "primp", "selenium", "ddgs", "requests")

Examples

```
## Not run:
response <- run_agent("Who is the president of France?", agent)
extracted <- extract_agent_results(response$trace)
print(extracted$search_snippets)
print(extracted$search_tiers)  # Shows which search tier was used

## End(Not run)
```

`extract_search_snippets` *Extract Search Snippets by Source Number*

Description

Extracts content from Search tool messages in the agent trace.

Usage

```
extract_search_snippets(text)
```

Arguments

`text`	Raw agent trace text

Value

Character vector of search snippets, ordered by source number

Examples

```
## Not run:
snippets <- extract_search_snippets(response$trace)

## End(Not run)
```

`extract_search_tiers` *Extract Search Tier Information*

Description

Extracts which search tier was used from the agent trace. The search module uses a multi-tier fallback system:

- primp: Fast HTTP client with browser impersonation (Tier 0)
- selenium: Headless browser for JS-rendered content (Tier 1)
- ddgs: Standard DDGS Python library (Tier 2)
- requests: Raw POST to DuckDuckGo HTML endpoint (Tier 3)

Usage

```
extract_search_tiers(text)
```

Arguments

`text`	Raw agent trace text

Value

Character vector of unique tier names encountered (e.g., "primp", "selenium", "ddgs", "requests")

Examples

```
## Not run:
tiers <- extract_search_tiers(response$trace)
print(tiers)  # e.g., "primp"

## End(Not run)
```

`extract_urls` *Extract URLs by Source Number*

Description

Extracts URLs from Search tool messages in the agent trace.

Usage

```
extract_urls(text)
```

Arguments

`text`	Raw agent trace text

Value

Character vector of URLs, ordered by source number

Examples

```
## Not run:
urls <- extract_urls(response$trace)

## End(Not run)
```

`extract_wikipedia_content` *Extract Wikipedia Content*

Description

Extracts content from Wikipedia tool messages in the agent trace.

Usage

```
extract_wikipedia_content(text)
```

Arguments

`text`	Raw agent trace text

Value

Character vector of Wikipedia snippets

Examples

```
## Not run:
wiki <- extract_wikipedia_content(response$trace)

## End(Not run)
```

`get_agent` *Get the Current Agent*

Description

Returns the currently initialized agent, or NULL if not initialized.

Usage

```
get_agent()
```

Value

An asa_agent object or NULL

Examples

```
## Not run:
agent <- get_agent()
if (is.null(agent)) {
  agent <- initialize_agent()
}

## End(Not run)
```

`get_tor_ip` *Get External IP via Tor*

Description

Retrieves the external IP address as seen through Tor proxy.

Usage

```
get_tor_ip(proxy = "socks5h://127.0.0.1:9050")
```

Arguments

proxy	Tor proxy URL

Value

IP address string or NA on failure

Examples

```
## Not run:
ip <- get_tor_ip()
message("Current Tor IP: ", ip)

## End(Not run)
```

`initialize_agent` *Initialize the ASA Search Agent*

Description

Initializes the Python environment and creates the LangGraph agent with search tools (Wikipedia, DuckDuckGo). The agent can use multiple LLM backends and supports DeepAgent-style memory folding.

Usage

```
initialize_agent(
  backend = "openai",
  model = "gpt-4.1-mini",
  conda_env = "asa_env",
  proxy = "socks5h://127.0.0.1:9050",
  use_memory_folding = TRUE,
  memory_threshold = 4L,
  memory_keep_recent = 2L,
  rate_limit = 0.2,
  timeout = 120L,
  verbose = TRUE
)
```

Arguments

`backend`	LLM backend to use. One of: "openai", "groq", "xai", "exo", "openrouter"
`model`	Model identifier (e.g., "gpt-4.1-mini", "llama-3.3-70b-versatile")
`conda_env`	Name of the conda environment with Python dependencies
`proxy`	SOCKS5 proxy URL for Tor (default: "socks5h://127.0.0.1:9050"). Set to NULL to disable proxy.
`use_memory_folding`	Enable DeepAgent-style memory compression (default: TRUE)
`memory_threshold`	Number of messages before folding triggers (default: 4)
`memory_keep_recent`	Number of recent messages to preserve after folding (default: 2)
`rate_limit`	Requests per second for rate limiting (default: 0.2)
`timeout`	Request timeout in seconds (default: 120)
`verbose`	Print status messages (default: TRUE)

Details

The agent is created with two tools:

- Wikipedia: For looking up encyclopedic information
- DuckDuckGo Search: For web searches with a 4-tier fallback system (PRIMP -> Selenium -> DDGS library -> raw requests)

Memory folding (enabled by default) compresses older messages into a summary to manage context length in long conversations, following the DeepAgent paper.

Value

An object of class `asa_agent` containing the initialized agent and configuration.

API Keys

The following environment variables should be set based on your backend:

- OpenAI: `OPENAI_API_KEY`
- Groq: `GROQ_API_KEY`
- xAI: `XAI_API_KEY`
- OpenRouter: `OPENROUTER_API_KEY`

OpenRouter Models

When using the `"openrouter"` backend, model names must be in `provider/model-name` format. Examples:

- `"openai/gpt-4o"`
- `"anthropic/claude-3-sonnet"`
- `"google/gemma-2-9b-it:free"`
- `"meta-llama/llama-3-70b-instruct"`

See `https://openrouter.ai/models` for available models.

See Also

run_task, run_task_batch

Examples

```
## Not run:
# Initialize with OpenAI
agent <- initialize_agent(
  backend = "openai",
  model = "gpt-4.1-mini"
)

# Initialize with Groq and custom settings
agent <- initialize_agent(
  backend = "groq",
  model = "llama-3.3-70b-versatile",
  use_memory_folding = FALSE,
  proxy = NULL  # No Tor proxy
)

# Initialize with OpenRouter (access to 100+ models)
agent <- initialize_agent(
  backend = "openrouter",
  model = "anthropic/claude-3-sonnet"  # Note: provider/model format
)

## End(Not run)
```

is_tor_running *Check if Tor is Running*

Description

Checks if Tor is running and accessible on the default port.

Usage

```
is_tor_running(port = 9050L)
```

Arguments

port	Port number (default: 9050)

Value

Logical indicating if Tor appears to be running

Examples

```
## Not run:
if (!is_tor_running()) {
  message("Start Tor with: brew services start tor")
}

## End(Not run)
```

`print.asa_agent` *Print Method for asa_agent Objects*

Description

Print Method for asa_agent Objects

Usage

```
## S3 method for class 'asa_agent'
print(x, ...)
```

Arguments

`x`	An asa_agent object
`...`	Additional arguments (ignored)

Value

Invisibly returns the object

`print.asa_audit_result` *Print Method for asa_audit_result Objects*

Description

Print Method for asa_audit_result Objects

Usage

```
## S3 method for class 'asa_audit_result'
print(x, n = 6, ...)
```

Arguments

`x`	An asa_audit_result object
`n`	Number of data rows to preview (default: 6)
`...`	Additional arguments (ignored)

Value

Invisibly returns the object

`print.asa_config` *Print Method for asa_config Objects*

Description

Print Method for asa_config Objects

Usage

```
## S3 method for class 'asa_config'
print(x, ...)
```

Arguments

`x`	An asa_config object
`...`	Additional arguments (ignored)

Value

Invisibly returns the object

`print.asa_enumerate_result` *Print Method for asa_enumerate_result Objects*

Description

Print Method for asa_enumerate_result Objects

Usage

```
## S3 method for class 'asa_enumerate_result'
print(x, n = 6, ...)
```

Arguments

`x`	An asa_enumerate_result object
`n`	Number of data rows to preview (default: 6)
`...`	Additional arguments (ignored)

Value

Invisibly returns the object

`print.asa_response` *Print Method for asa_response Objects*

Description

Print Method for asa_response Objects

Usage

```
## S3 method for class 'asa_response'
print(x, ...)
```

Arguments

`x`	An asa_response object
`...`	Additional arguments (ignored)

Value

Invisibly returns the object

`print.asa_result` *Print Method for asa_result Objects*

Description

Print Method for asa_result Objects

Usage

```
## S3 method for class 'asa_result'
print(x, ...)
```

Arguments

`x`	An asa_result object
`...`	Additional arguments (ignored)

Value

Invisibly returns the object

print.asa_search *Print Method for asa_search Objects*

Description

Print Method for asa_search Objects

Usage

```
## S3 method for class 'asa_search'
print(x, ...)
```

Arguments

x	An asa_search object
...	Additional arguments (ignored)

print.asa_temporal *Print Method for asa_temporal Objects*

Description

Print Method for asa_temporal Objects

Usage

```
## S3 method for class 'asa_temporal'
print(x, ...)
```

Arguments

x	An asa_temporal object
...	Additional arguments (ignored)

Value

Invisibly returns the object

`process_outputs` *Process Multiple Agent Outputs*

Description

Processes a data frame of raw agent outputs, extracting structured data.

Usage

```
process_outputs(df, parallel = FALSE, workers = 10L)
```

Arguments

`df`	Data frame with a 'raw_output' column containing agent traces
`parallel`	Use parallel processing
`workers`	Number of workers

Value

The input data frame with additional extracted columns: search_count, wiki_count, and any JSON fields found

`reset_agent` *Reset the Agent*

Description

Clears the initialized agent state, forcing reinitialization on next use. Also closes any open HTTP clients to prevent resource leaks.

Usage

```
reset_agent()
```

Value

Invisibly returns NULL

`rotate_tor_circuit` *Rotate Tor Circuit*

Description

Requests a new Tor circuit by restarting the Tor service.

Usage

```
rotate_tor_circuit(method = c("brew", "systemctl", "signal"), wait = 12L)
```

Arguments

`method`	Method to restart: "brew" (macOS), "systemctl" (Linux), or "signal"
`wait`	Seconds to wait for new circuit (default: 12)

Value

Invisibly returns NULL

Examples

```
## Not run:
rotate_tor_circuit()

## End(Not run)
```

`run_task` *Run a Structured Task with the Agent*

Description

Executes a research task using the AI search agent with a structured prompt and returns parsed results. This is the primary function for running agent tasks.

Usage

```
run_task(
  prompt,
  output_format = "text",
  temporal = NULL,
  config = NULL,
  agent = NULL,
  verbose = FALSE
)
```

Arguments

`prompt`	The task prompt or question for the agent to research
`output_format`	Expected output format. One of: • "text": Returns response text (default) • "json": Parse response as JSON • "raw": Include full trace in result for debugging • Character vector: Extract specific fields from response
`temporal`	Named list or `asa_temporal` object for temporal filtering: • time_filter: DuckDuckGo time filter - "d" (day), "w" (week), "m" (month), "y" (year) • after: ISO 8601 date (e.g., "2020-01-01") - hint for results after this date (added to prompt context) • before: ISO 8601 date (e.g., "2024-01-01") - hint for results before this date (added to prompt context)
`config`	An `asa_config` object for unified configuration, or NULL to use defaults
`agent`	An asa_agent object from `initialize_agent`, or NULL to use the currently initialized agent
`verbose`	Print progress messages (default: FALSE)

Details

This function provides the primary interface for running research tasks. For simple text responses, use `output_format = "text"`. For structured outputs, use `output_format = "json"` or specify field names to extract. For debugging and full trace access, use `output_format = "raw"`.

When temporal filtering is specified, the search tool's time filter is temporarily set for this task and restored afterward. Date hints (after/before) are appended to the prompt to guide the agent's search behavior.

Value

An `asa_result` object with:

- prompt: The original prompt
- message: The agent's response text
- parsed: Parsed output (list for JSON/field extraction, NULL for text/raw)
- raw_output: Full agent trace (always included, verbose for "raw" format)
- elapsed_time: Execution time in minutes
- status: "success" or "error"
- trace: Full execution trace (for "raw" output_format)
- fold_count: Number of memory folds (for "raw" output_format)

See Also

`initialize_agent`, `run_task_batch`, `asa_config`, `temporal_options`

Examples

```
## Not run:
# Initialize agent first
agent <- initialize_agent(backend = "openai", model = "gpt-4.1-mini")

# Simple text query
result <- run_task(
  prompt = "What is the capital of France?",
  output_format = "text",
  agent = agent
)
print(result$message)

# JSON structured output
result <- run_task(
  prompt = "Find information about Albert Einstein and return JSON with
            fields: birth_year, death_year, nationality, field_of_study",
  output_format = "json",
  agent = agent
)
print(result$parsed)

# Raw output for debugging (includes full trace in asa_result)
result <- run_task(
  prompt = "Search for information",
  output_format = "raw",
  agent = agent
)
cat(result$trace)  # View full agent trace

# With temporal filtering (past year only)
result <- run_task(
  prompt = "Find recent AI research breakthroughs",
  temporal = temporal_options(time_filter = "y"),
  agent = agent
)

# With date range hint
result <- run_task(
  prompt = "Find tech companies founded recently",
  temporal = list(
    time_filter = "y",
    after = "2020-01-01",
    before = "2024-01-01"
  ),
  agent = agent
)

# Using asa_config for unified configuration
config <- asa_config(
  backend = "openai",
  model = "gpt-4.1-mini",
  temporal = temporal_options(time_filter = "y")
)
result <- run_task(prompt, config = config)
```

```
## End(Not run)
```

run_task_batch *Run Multiple Tasks in Batch*

Description

Executes multiple research tasks, optionally in parallel.

Usage

```
run_task_batch(
  prompts,
  output_format = "text",
  temporal = NULL,
  agent = NULL,
  parallel = FALSE,
  workers = 4L,
  progress = TRUE
)
```

Arguments

`prompts`	Character vector of task prompts, or a data frame with a 'prompt' column
`output_format`	Expected output format (applies to all tasks)
`temporal`	Named list for temporal filtering (applies to all tasks). See `run_task` for details.
`agent`	An asa_agent object
`parallel`	Use parallel processing
`workers`	Number of parallel workers
`progress`	Show progress messages

Value

A list of asa_result objects, or if prompts was a data frame, the data frame with result columns added

See Also

`run_task`, `configure_temporal`

Examples

```
## Not run:
prompts <- c(
  "What is the population of Tokyo?",
  "What is the population of New York?",
  "What is the population of London?"
)
results <- run_task_batch(prompts, agent = agent)
```

```
# With temporal filtering for all tasks
results <- run_task_batch(
  prompts,
  temporal = list(time_filter = "y"),
  agent = agent
)

## End(Not run)
```

`search_options` *Create Search Options*

Description

Creates search configuration for controlling DuckDuckGo search behavior, including rate limiting, retry policies, and result limits. These options are used by the 4-tier search fallback system.

Usage

```
search_options(
  max_results = NULL,
  timeout = NULL,
  max_retries = NULL,
  retry_delay = NULL,
  backoff_multiplier = NULL,
  inter_search_delay = NULL
)
```

Arguments

`max_results`	Maximum number of search results to return per query. Higher values provide more context but increase latency. Default: 10.
`timeout`	Timeout in seconds for individual search requests. Applies to each tier attempt separately. Default: 15.
`max_retries`	Maximum number of retry attempts when a search tier fails. After exhausting retries, the system falls back to the next tier. Default: 3.
`retry_delay`	Initial delay in seconds before the first retry. Subsequent retries use exponential backoff. Default: 2.
`backoff_multiplier`	Multiplier for exponential backoff between retries. E.g., with retry_delay=2 and multiplier=1.5, delays are 2s, 3s, 4.5s. Default: 1.5.
`inter_search_delay`	Minimum delay in seconds between consecutive searches. Helps avoid rate limiting from search providers. Default: 0.5.

Details

The search system uses a 4-tier fallback architecture:

1. **PRIMP**: HTTP/2 with browser TLS fingerprint
2. **Selenium**: Headless browser for JS-rendered content
3. **DDGS**: Standard ddgs Python library
4. **Requests**: Raw POST to DuckDuckGo HTML endpoint

The retry/backoff settings apply within each tier. If all retries are exhausted, the system automatically falls back to the next tier.

Value

An object of class `asa_search`

See Also

`asa_config`, `configure_search`

Examples

```
## Not run:
# Default settings
search <- search_options()

# More aggressive settings for faster searches
search <- search_options(
  max_results = 5,
  timeout = 10,
  max_retries = 2
)

# Conservative settings for rate-limited environments
search <- search_options(
  inter_search_delay = 2.0,
  max_retries = 5,
  backoff_multiplier = 2.0
)

# Use with asa_config
config <- asa_config(
  backend = "openai",
  search = search_options(max_results = 15)
)

## End(Not run)
```

`summary.asa_agent` *Summary Method for asa_agent Objects*

Description

Summary Method for asa_agent Objects

Usage

```
## S3 method for class 'asa_agent'
summary(object, ...)
```

Arguments

`object`	An asa_agent object
`...`	Additional arguments (ignored)

Value

Invisibly returns a summary list

`summary.asa_audit_result` *Summary Method for asa_audit_result Objects*

Description

Summary Method for asa_audit_result Objects

Usage

```
## S3 method for class 'asa_audit_result'
summary(object, ...)
```

Arguments

`object`	An asa_audit_result object
`...`	Additional arguments (ignored)

Value

Invisibly returns a summary list

summary.asa_enumerate_result *Summary Method for asa_enumerate_result Objects*

Description

Summary Method for asa_enumerate_result Objects

Usage

```
## S3 method for class 'asa_enumerate_result'
summary(object, ...)
```

Arguments

`object`	An asa_enumerate_result object
`...`	Additional arguments (ignored)

Value

Invisibly returns a summary list

summary.asa_response *Summary Method for asa_response Objects*

Description

Summary Method for asa_response Objects

Usage

```
## S3 method for class 'asa_response'
summary(object, show_trace = FALSE, ...)
```

Arguments

`object`	An asa_response object
`show_trace`	Include full trace in output
`...`	Additional arguments (ignored)

Value

Invisibly returns a summary list

summary.asa_result *Summary Method for asa_result Objects*

Description

Summary Method for asa_result Objects

Usage

```
## S3 method for class 'asa_result'
summary(object, ...)
```

Arguments

object	An asa_result object
...	Additional arguments (ignored)

Value

Invisibly returns a summary list

temporal_options *Create Temporal Filtering Options*

Description

Creates a temporal filtering configuration for constraining search results by date. Supports DuckDuckGo time filters, date ranges, and strict verification modes.

Usage

```
temporal_options(
  time_filter = NULL,
  after = NULL,
  before = NULL,
  strictness = "best_effort",
  use_wayback = FALSE
)
```

Arguments

time_filter	DuckDuckGo time filter: "d" (day), "w" (week), "m" (month), "y" (year), or NULL for no filter
after	ISO 8601 date string (e.g., "2020-01-01") - results after this date
before	ISO 8601 date string (e.g., "2024-01-01") - results before this date
strictness	Verification level: "best_effort" (default) or "strict"
use_wayback	Use Wayback Machine for strict pre-date guarantees

Details

Temporal filtering can operate at different levels:

- **time_filter**: DuckDuckGo native filter (fast, approximate)
- **after/before**: Date hints appended to prompts
- **strict**: Post-hoc verification of result dates
- **use_wayback**: Uses Internet Archive for guaranteed historical data

Value

An object of class `asa_temporal`

See Also

`asa_config`, `run_task`

Examples

```
## Not run:
# Past year only
temporal <- temporal_options(time_filter = "y")

# Specific date range
temporal <- temporal_options(
  after = "2020-01-01",
  before = "2024-01-01"
)

# Strict historical verification
temporal <- temporal_options(
  before = "2015-01-01",
  strictness = "strict",
  use_wayback = TRUE
)

## End(Not run)
```

`write_csv.asa_enumerate_result` *Write asa_enumerate_result to CSV*

Description

Write asa_enumerate_result to CSV

Usage

```
write_csv.asa_enumerate_result(x, file, include_provenance = FALSE, ...)
```

Arguments

`x`	An asa_enumerate_result object
`file`	Path to output CSV file
`include_provenance`	Include provenance as additional columns
`...`	Additional arguments passed to write.csv

Value

Invisibly returns the file path

Index